**组成原理实验课程第 四 次实报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | ALU 模块实现 | | | 班级 | 李涛 |
| 学生姓名 | 艾明旭 | 学号 | 2111033 | 指导老师 | 董前琨 |
| 实验地点 | B306 | | 实验时间 | 2023年4月25日 | |

1. **实验目的**
2. 熟悉MIPS指令集中的运算指令，学会对这些指令进行归纳分类。
3. 了解MIPS指令结构。
4. 熟悉并掌握ALU的原理、功能和设计。
5. 进一步加强运用verilog语言进行电路设计的能力。
6. 为后续设计cpu的实验打下基础。

**实验设备**

1. 装有Xilinx Vivado的计算机一台。
2. LS-CPU-EXB-002教学系统实验箱一套。
3. **实验内容说明**

1.学习 MIPS 指令集，熟知指令类型，了解指令功能和编码，归纳基础的 ALU 运算指令。

2.归纳确定自己本次实验中准备实现的 ALU 运算，要求不实现定点乘除指令和浮点运算指令，要求至少实现 5 种 ALU 运算，其中要包含加减运算，其中减法在内部要转换为加法，与加法运算共同调用实验一里自己完成的加法模块去做。

3.自行设计本次实验的方案，画出结构框图，大致结构框图如图 4.1。图 4.1 中的操作码位数和类型请自行设计，可以设计为独热码（一位有效编码）或二进制编码。比如，设计方案中预定实现 7 种 ALU 运算，则操作码采用独热码，则需 7bit 数据，每位单独指示一种运算；若采用二进制编码，则只用 3bit 数据位即可，但在需 ALU 内部先进行解码，才能确定 ALU 作何种运算

4.根据设计的实验方案，使用verilog编写相应代码。

5.对编写的代码进行仿真，得到正确的波形图。

6.将以上设计作为一个单独的模块，设计一个外围模块去调用该模块，见图5.2。外围模块中需调用封装好的LCD触摸屏模块，显示ALU的两个源操作数、操作码和运算结果，并且需要利用触摸功能输入源操作数。操作码可以考虑用LCD触摸屏输入，也可以用拨码开关输入。

7.将编写的代码进行综合布局布线，并下载到试验箱中的 FPGA 板子上进行演示。

8.ALU 运算器改进要求

1. **实验原理图**

图 5.1 ALU模块的大致框图

32位源操作数1

32位源操作数2

运算结果

操作码

外围模块

来自FPGA板子上的输入

输出到FPGA板上进行展示

图 5.2 ALU设计实验的顶层模块大致框图



### 2.2 对原始 ALU 代码进行修改

• 压缩 ALU 运算器的符号控制独热码至 4 位

• 增加三个自定义运算加入 ALU 中

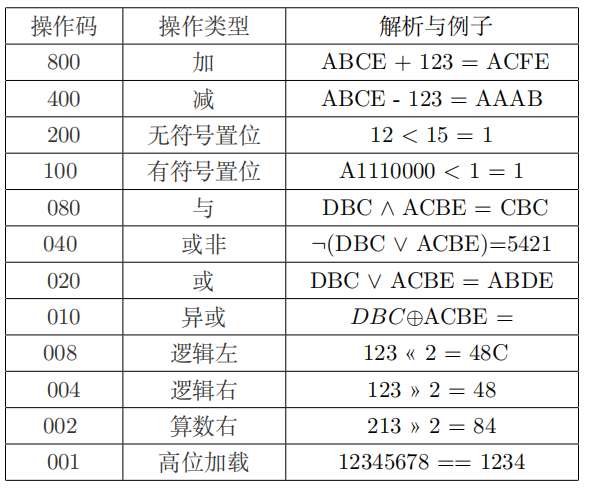
按位取反

与非

大于置位

**4.实验步骤**

1.初始计算表格



1修改过后的结果为：

|  |  |  |
| --- | --- | --- |
| 001 | 高位加载 | 12345678=1234 |
| 002 | 算数右 | 213>>2=84 |
| 003 | 逻辑右 | 123>>2=48 |
| 004 | 逻辑左 | 123<<2=48C |
| 005 | 异或 | 12*⊕ 22=30* |
| 006 | 或 | DBC ∨ ACBE = ABDE |
| 007 | 或非 | ¬(DBC ∨ ACBE)=5421 |
| 008 | 与 | DBC ∧ ACBE = CBC |
| 009 | 有符号置位 | ¬(DBC ∨ ACBE)=5421 |
| 00A | 无符号置位 | DBC ∨ ACBE = ABDE |
| 00B | 减 | ABCE - 123 = AAAB |
| 00C | 加 | ABCE + 123 = ACFE |
| 00D | 按位取反 | ¬00=FF |
| 00E | 与非 | ¬（0∧0）=1 |
| 00F | 大于则置位（无符号） | ACBE ∨ DBC = ABDE |

**4.2 上述表格 ALU 操作代码原理**

• 加减法

利用之前实验的 adder

• 置位操作

使用 adder 计算两个操作数的差，若为有符号数，根据差的符号位和两个操作数的符号位，

由此化简真值表得到结果; 对于 32 无符号位比较，在其最高位前填 0 作为 33 位正数比较。

• 逻辑操作

按位与、或、非、异或等。

• 左、右移

根据移动位数来按位左移和右移几位，注意算数左移和逻辑右移实际是一样的

• 高低位加载

读取指定操作数的高、低 16 位的值，使用 veilog 的切片操作即可。

**4.3 代码修改**

1. 压缩控制信号至 4 位

module alu(

input [3:0] alu\_control, // ALU控制信号,改为4位

input [31:0] alu\_src1, // ALU操作数1,为补码

input [31:0] alu\_src2, // ALU操作数2，为补码

output [31:0] alu\_result // ALU结果

);

2.根据 4 位控制信号进行独热编码，且增加三个独热码

• 增加独热码

wire alu\_not; //按位取反

wire alu\_nand; //按位与非

wire alu\_sgt; //有符号比较，大于置位，复用加法器做减法(验证比较--两种运算）

4 位控制信号转为独热码

assign alu\_add = alu\_control==4'b0001;

assign alu\_sub = alu\_control==4'b0010;

assign alu\_slt = alu\_control==4'b0011;

assign alu\_sltu = alu\_control==4'b0100;

assign alu\_and = alu\_control==4'b0101;

assign alu\_nor = alu\_control==4'b0110;

assign alu\_or = alu\_control==4'b0111;

assign alu\_xor = alu\_control==4'b1000;

assign alu\_sll = alu\_control==4'b1001;

assign alu\_srl = alu\_control==4'b1010;

assign alu\_sra = alu\_control==4'b1011;

assign alu\_lui = alu\_control==4'b1100;

assign alu\_not = alu\_control==4'b1101;

assign alu\_nand = alu\_control==4'b1110;

assign alu\_sgt = alu\_control==4'b1111;

3. 增加的三个操作代码

按位取反，与非和大于置位

assign not\_result = ~ alu\_src2; //按位取反操作对源操作数2进行按位取反

assign nand\_result = ~and\_result; //与非结果为与结果取反

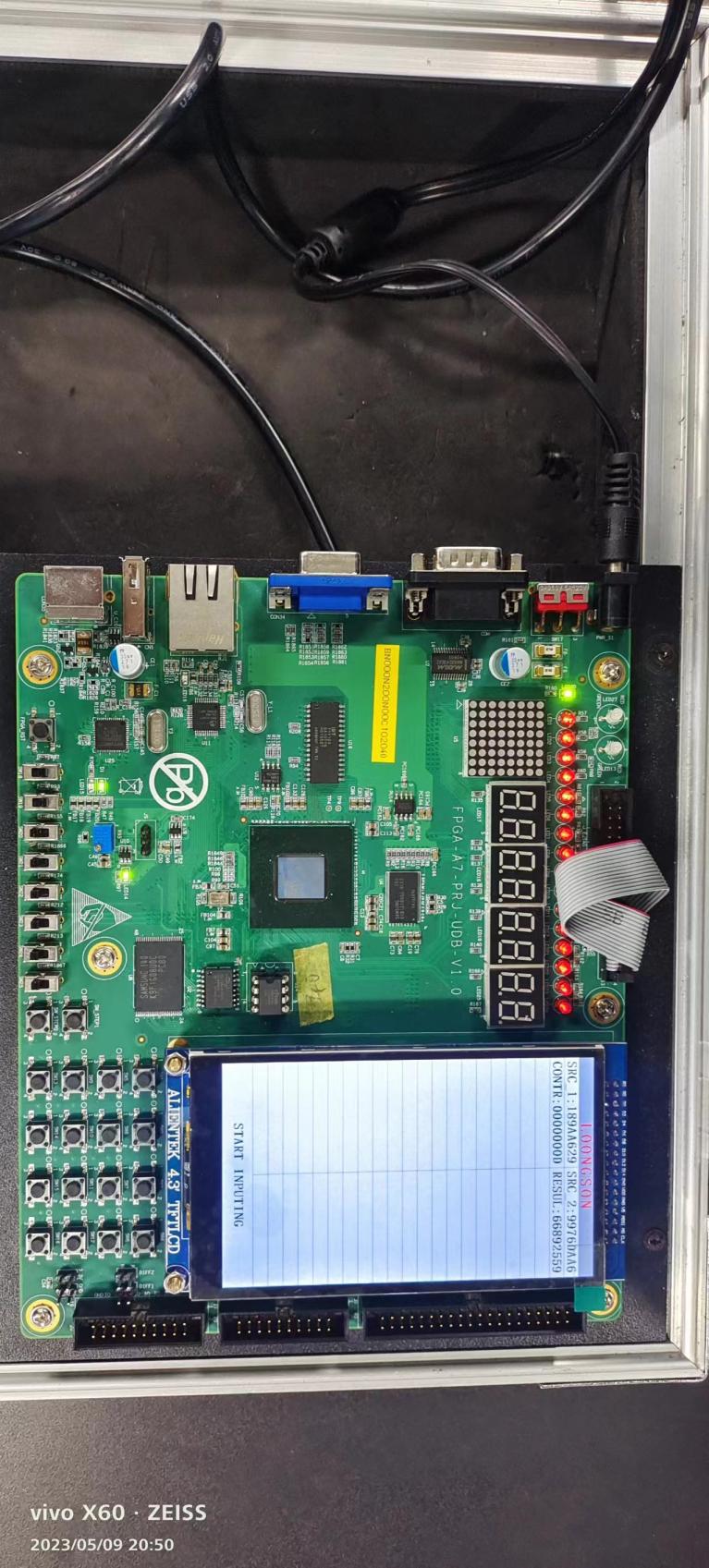
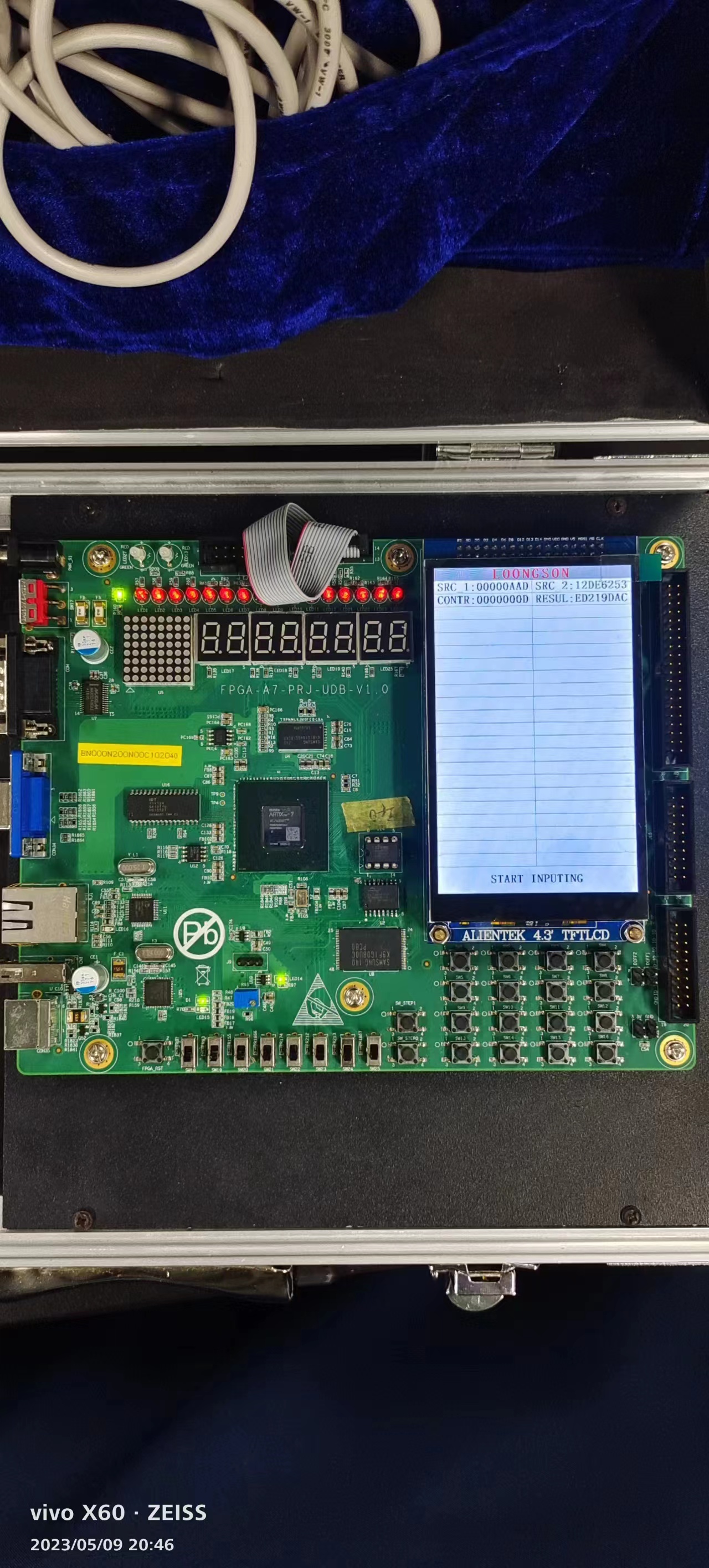
assign sgt\_result[31:1] = 31'd0;

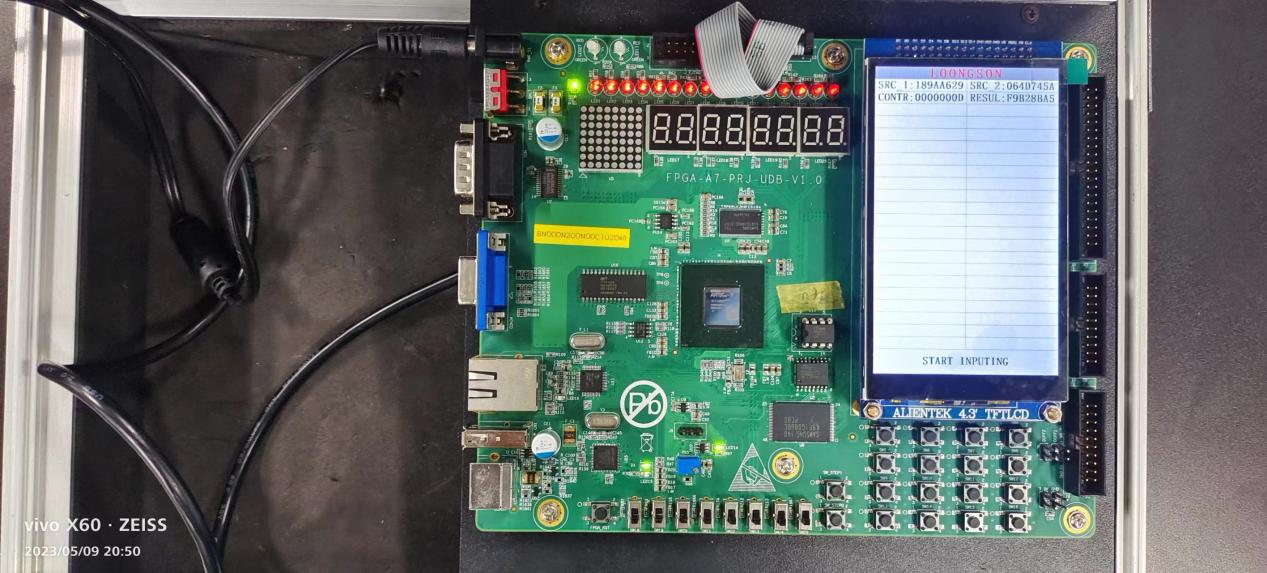
assign sgt\_result[0] = ~((alu\_src1[31] & ~alu\_src2[31]) | (~(alu\_src1[31]^alu\_src2[31]) & adder\_result[31]))

& (|(alu\_src1-alu\_src2)); //对slt的判断按位取反后再去掉两个操作数相等的情况

**5 实验结果分析**

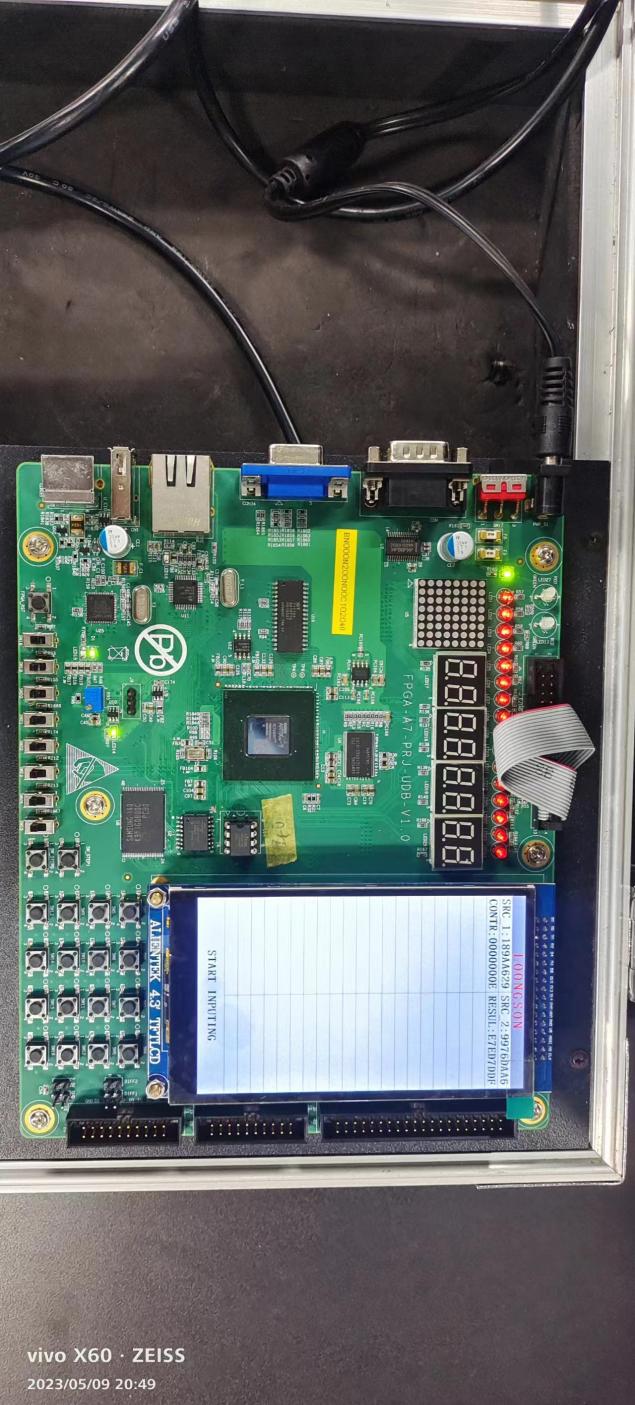
1.按位取反的结果，对于src2的数据进行按位取反的计算

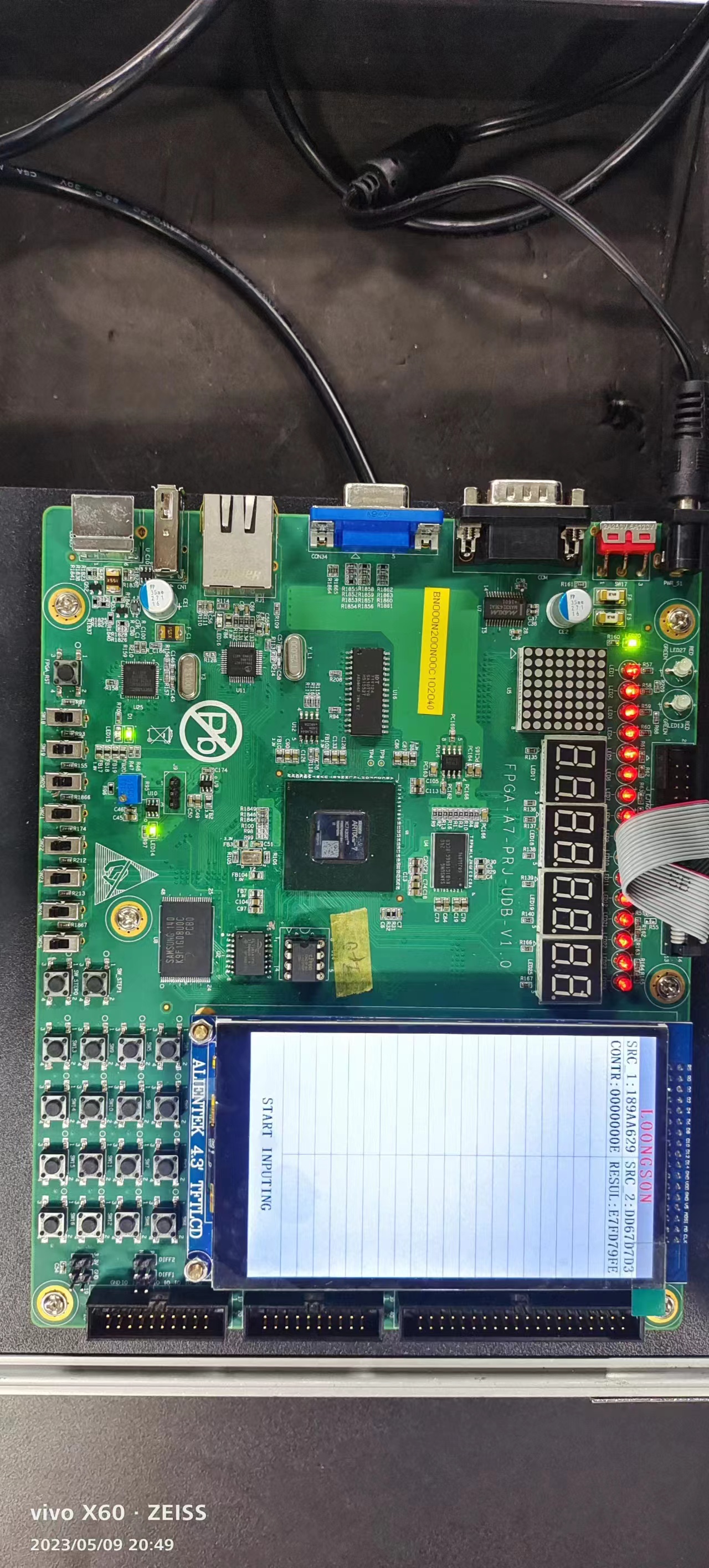


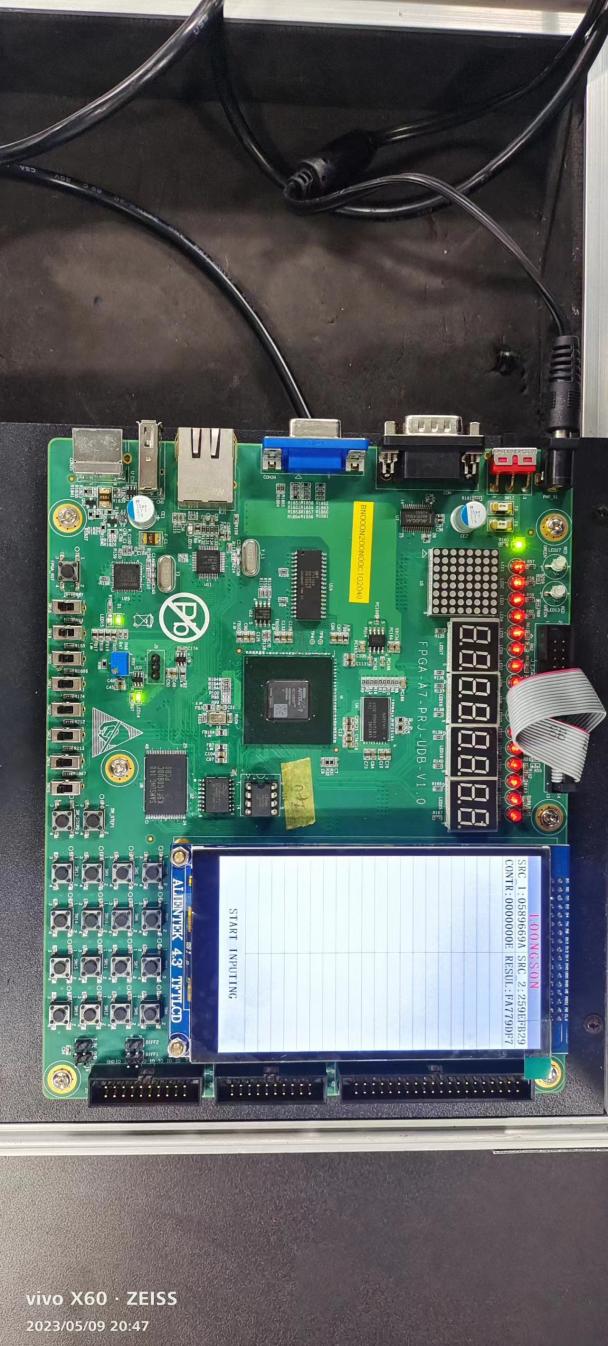


2.按位与非的结果

该位全0为1，否则为0

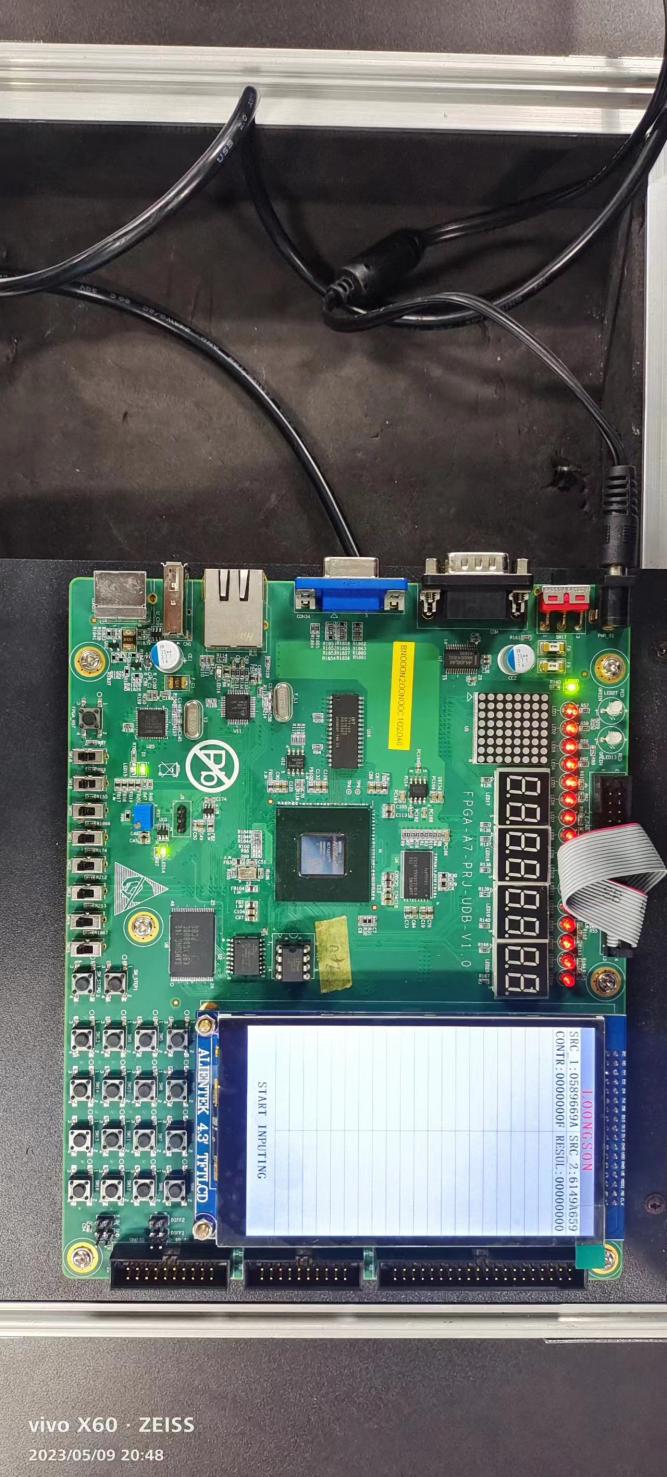


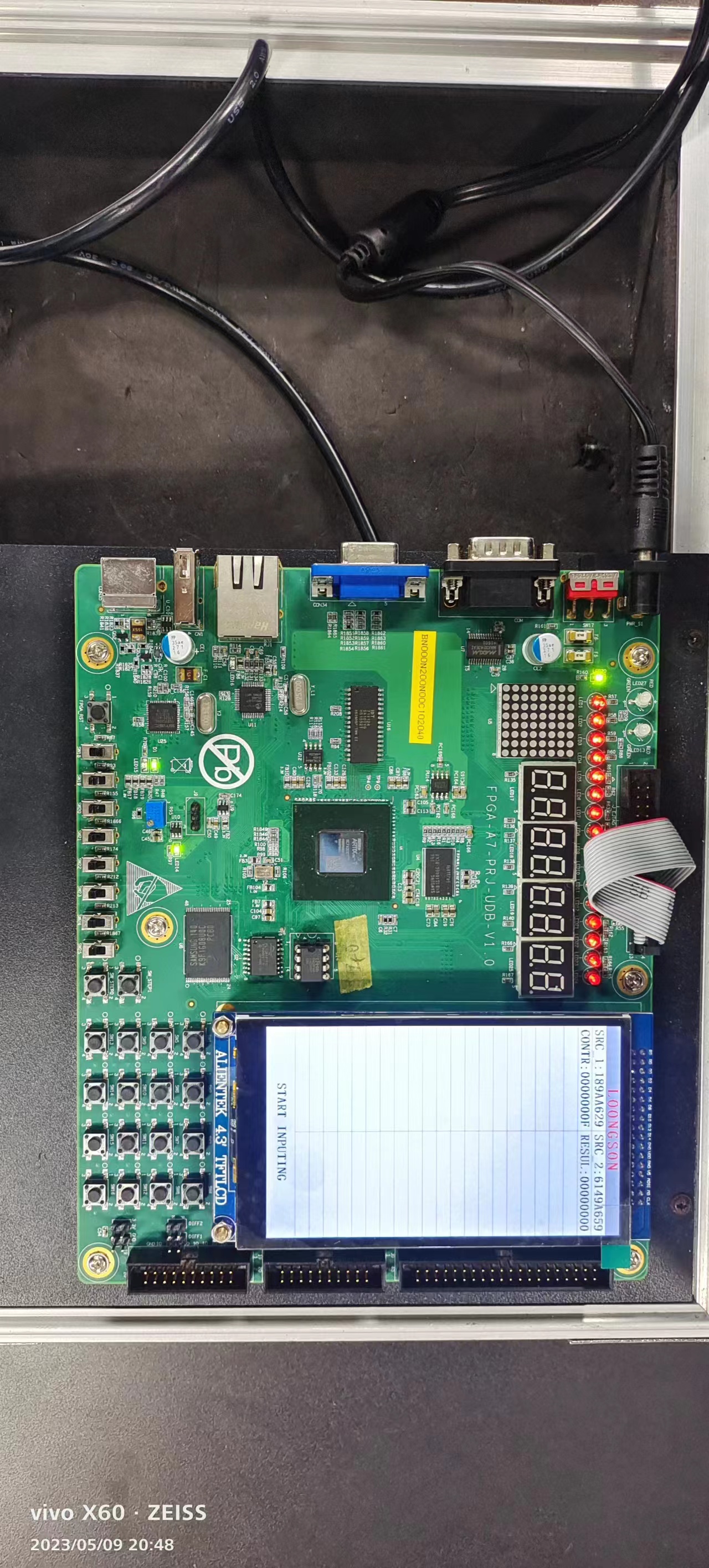


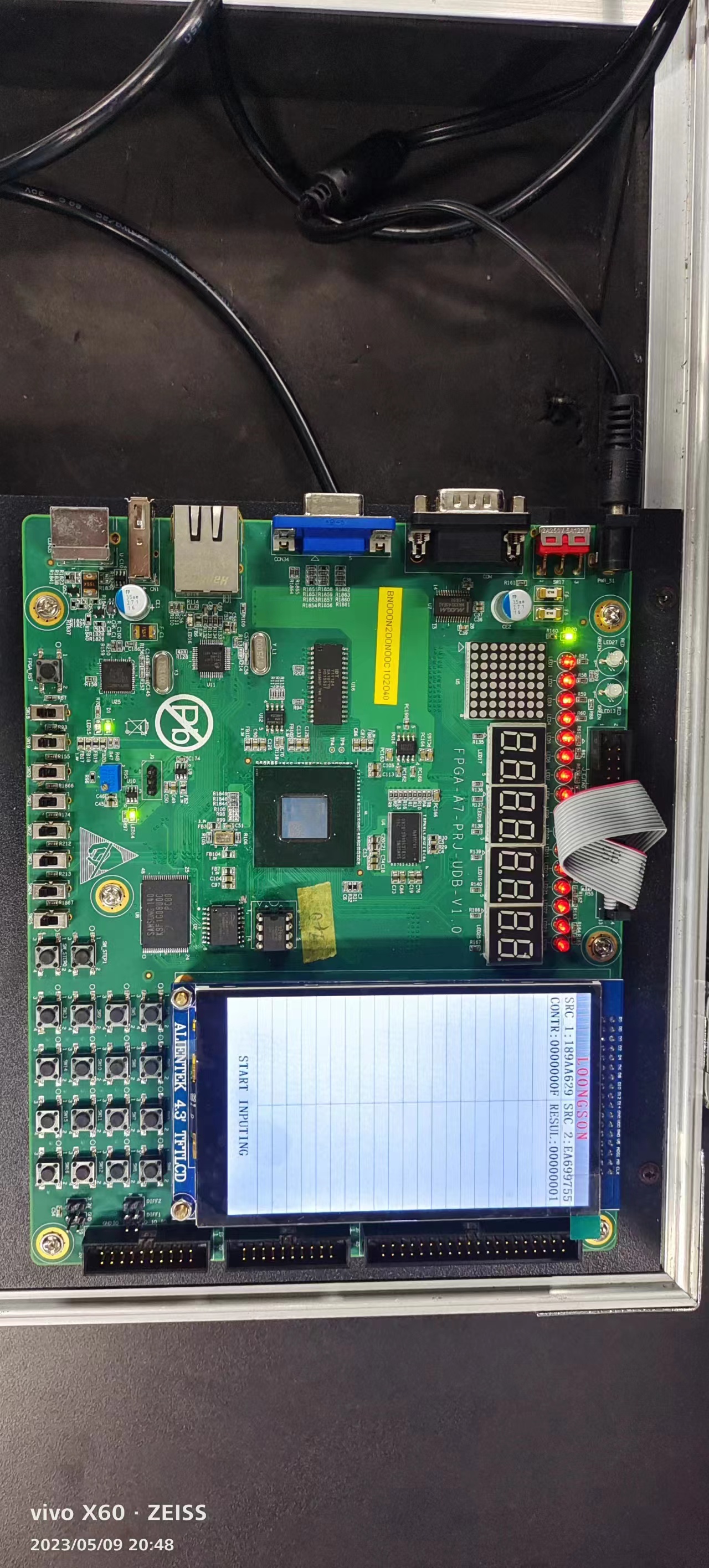


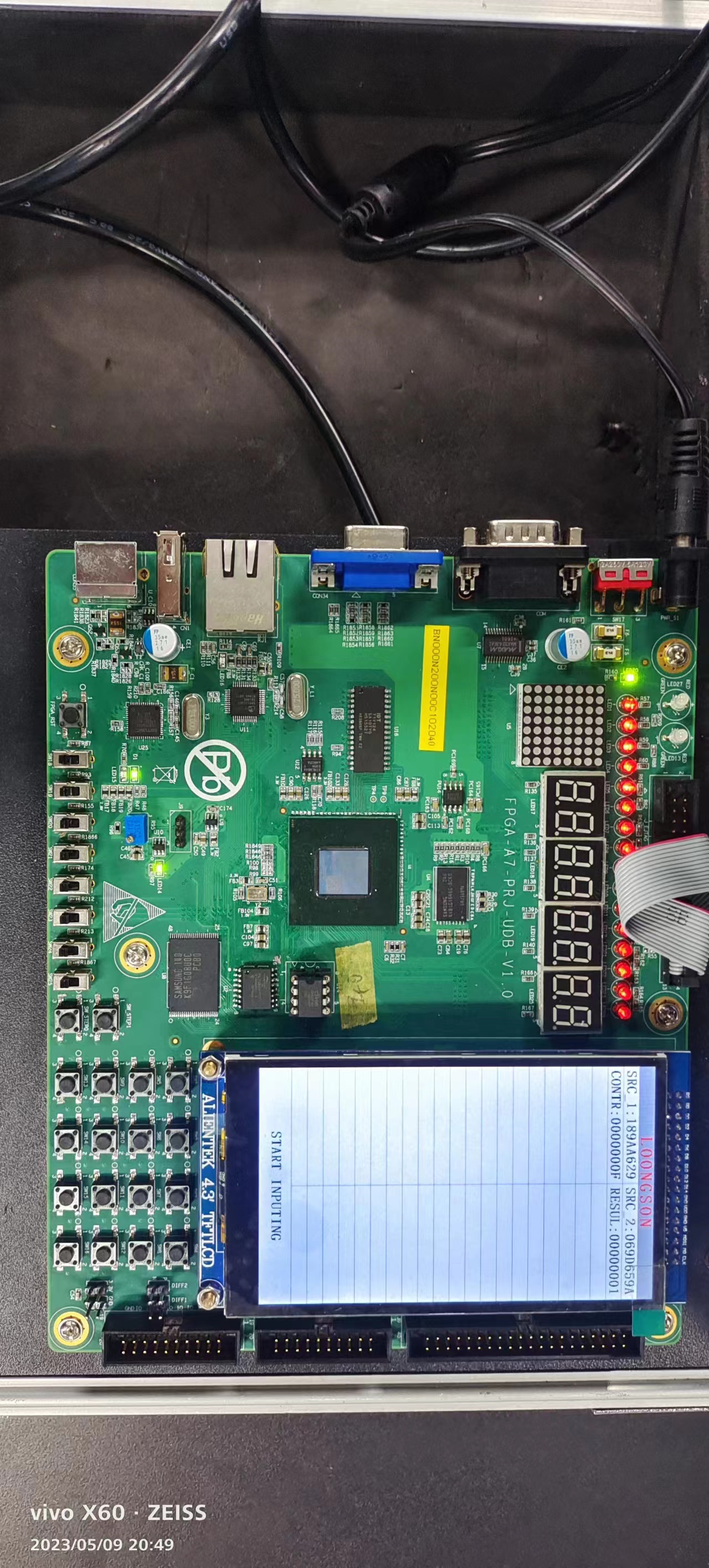
3.大于则置位的计算结果

无符号数组 A 大于 b，则大于置位 1









**6.总结感想**

这次实验，我学习了 ALU 应对不同指令时内部的工作原理，对于不同的指令会产生对于的控制码，而 ALU 会产生所有操作类型的结果，但最后只输出控制码对应的结果。也使得我对 verilog 语言更加熟悉，能够熟练上机仿真。

同时，我还学会了利用两位移位进行的数据乘算和移位操作，还对数据的末尾对数据的乘法运算的影响有了一定的了解。

最后，这次实验还很好的为数据通路和 CPU 设计的学习有了辅助和铺垫，为后续的学习打好良好的基础。

附录：全部代码

Alu.v

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// > 文件名: alu.v

// > 描述 ：ALU模块，可做12种操作

// > 作者 : LOONGSON

// > 日期 : 2016-04-14

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module alu(

input [3:0] alu\_control, // ALU控制信号,改为4位

input [31:0] alu\_src1, // ALU操作数1,为补码

input [31:0] alu\_src2, // ALU操作数2，为补码

output [31:0] alu\_result // ALU结果

);

// ALU控制信号，独热码

wire alu\_add; //加法操作

wire alu\_sub; //减法操作

wire alu\_slt; //有符号比较，小于置位，复用加法器做减法

wire alu\_sltu; //无符号比较，小于置位，复用加法器做减法

wire alu\_and; //按位与

wire alu\_nor; //按位或非

wire alu\_or; //按位或

wire alu\_xor; //按位异或

wire alu\_sll; //逻辑左移

wire alu\_srl; //逻辑右移

wire alu\_sra; //算术右移

wire alu\_lui; //高位加载

wire alu\_not; //按位取反

wire alu\_nand; //按位与非

wire alu\_sgt; //有符号比较，大于置位，复用加法器做减法(验证比较--两种运算）

assign alu\_add = alu\_control==4'b0001;

assign alu\_sub = alu\_control==4'b0010;

assign alu\_slt = alu\_control==4'b0011;

assign alu\_sltu = alu\_control==4'b0100;

assign alu\_and = alu\_control==4'b0101;

assign alu\_nor = alu\_control==4'b0110;

assign alu\_or = alu\_control==4'b0111;

assign alu\_xor = alu\_control==4'b1000;

assign alu\_sll = alu\_control==4'b1001;

assign alu\_srl = alu\_control==4'b1010;

assign alu\_sra = alu\_control==4'b1011;

assign alu\_lui = alu\_control==4'b1100;

assign alu\_not = alu\_control==4'b1101;

assign alu\_nand = alu\_control==4'b1110;

assign alu\_sgt = alu\_control==4'b1111;

wire [31:0] add\_sub\_result;

wire [31:0] slt\_result;

wire [31:0] sltu\_result;

wire [31:0] and\_result;

wire [31:0] nor\_result;

wire [31:0] or\_result;

wire [31:0] xor\_result;

wire [31:0] sll\_result;

wire [31:0] srl\_result;

wire [31:0] sra\_result;

wire [31:0] lui\_result;

wire [31:0] not\_result;

wire [31:0] nand\_result;

wire [31:0] sgt\_result;

assign and\_result = alu\_src1 & alu\_src2; // 与结果为两数按位与

assign or\_result = alu\_src1 | alu\_src2; // 或结果为两数按位或

assign nor\_result = ~or\_result; // 或非结果为或结果按位取反

assign xor\_result = alu\_src1 ^ alu\_src2; // 异或结果为两数按位异或

assign lui\_result = {alu\_src2[15:0], 16'd0}; // 立即数装载结果为立即数移位至高半字节

assign not\_result = ~ alu\_src2; //按位取反操作对源操作数2进行按位取反

assign nand\_result = ~and\_result; //与非结果为与结果取反

//-----{加法器}begin

//add,sub,slt,sltu均使用该模块

wire [31:0] adder\_operand1;

wire [31:0] adder\_operand2;

wire adder\_cin ;

wire [31:0] adder\_result ;

wire adder\_cout ;

assign adder\_operand1 = alu\_src1;

assign adder\_operand2 = alu\_add ? alu\_src2 : ~alu\_src2;

assign adder\_cin = ~alu\_add; //减法需要cin

adder adder\_module(

.operand1(adder\_operand1),

.operand2(adder\_operand2),

.cin (adder\_cin ),

.result (adder\_result ),

.cout (adder\_cout )

);

//加减结果

assign add\_sub\_result = adder\_result;

//slt结果

//adder\_src1[31] adder\_src2[31] adder\_result[31]

// 0 1 X(0或1) "正-负"，显然小于不成立

// 0 0 1 相减为负，说明小于

// 0 0 0 相减为正，说明不小于

// 1 1 1 相减为负，说明小于

// 1 1 0 相减为正，说明不小于

// 1 0 X(0或1) "负-正"，显然小于成立

assign slt\_result[31:1] = 31'd0;

assign slt\_result[0] = (alu\_src1[31] & ~alu\_src2[31]) | (~(alu\_src1[31]^alu\_src2[31]) & adder\_result[31]);

//sltu结果

//对于32位无符号数比较，相当于33位有符号数（{1'b0,src1}和{1'b0,src2}）的比较，最高位0为符号位

//故，可以用33位加法器来比较大小，需要对{1'b0,src2}取反,即需要{1'b0,src1}+{1'b1,~src2}+cin

//但此处用的为32位加法器，只做了运算: src1 + ~src2 +cin

//32位加法的结果为{adder\_cout,adder\_result},则33位加法结果应该为{adder\_cout+1'b1,adder\_result}

//对比slt结果注释，知道，此时判断大小属于第二三种情况，即源操作数1符号位为0，源操作数2符号位为0

//结果的符号位为1，说明小于，即adder\_cout+1'b1为2'b01，即adder\_cout为0

assign sltu\_result = {31'd0, ~adder\_cout};

//sgt结果

//adder\_src1[31] adder\_src2[31] adder\_result[31]

// 0 1 X(0或1) "正-负"，显然大于成立

// 0 0 1 相减为负，说明不大于（实际上小于）

// 0 0 0 相减为正（或0），说明大于或等于

// 1 1 1 相减为负，说明不大于（实际上小于）

// 1 1 0 相减为正（或0），说明大于或等于

// 1 0 X(0或1) "负-正"，显然大于不成立

assign sgt\_result[31:1] = 31'd0;

assign sgt\_result[0] = ~((alu\_src1[31] & ~alu\_src2[31]) | (~(alu\_src1[31]^alu\_src2[31]) & adder\_result[31]))

& (|(alu\_src1-alu\_src2)); //对slt的判断按位取反后再去掉两个操作数相等的情况

//-----{加法器}end

//-----{移位器}begin

// 移位分三步进行，

// 第一步根据移位量低2位即[1:0]位做第一次移位，

// 第二步在第一次移位基础上根据移位量[3:2]位做第二次移位，

// 第三步在第二次移位基础上根据移位量[4]位做第三次移位。

wire [4:0] shf;

assign shf = alu\_src1[4:0];

wire [1:0] shf\_1\_0;

wire [1:0] shf\_3\_2;

assign shf\_1\_0 = shf[1:0];

assign shf\_3\_2 = shf[3:2];

// 逻辑左移

wire [31:0] sll\_step1;

wire [31:0] sll\_step2;

assign sll\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[30:0], 1'd0} // 若shf[1:0]="01",左移1位

| {32{shf\_1\_0 == 2'b10}} & {alu\_src2[29:0], 2'd0} // 若shf[1:0]="10",左移2位

| {32{shf\_1\_0 == 2'b11}} & {alu\_src2[28:0], 3'd0}; // 若shf[1:0]="11",左移3位

assign sll\_step2 = {32{shf\_3\_2 == 2'b00}} & sll\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {sll\_step1[27:0], 4'd0} // 若shf[3:2]="01",第一次移位结果左移4位

| {32{shf\_3\_2 == 2'b10}} & {sll\_step1[23:0], 8'd0} // 若shf[3:2]="10",第一次移位结果左移8位

| {32{shf\_3\_2 == 2'b11}} & {sll\_step1[19:0], 12'd0}; // 若shf[3:2]="11",第一次移位结果左移12位

assign sll\_result = shf[4] ? {sll\_step2[15:0], 16'd0} : sll\_step2; // 若shf[4]="1",第二次移位结果左移16位

// 逻辑右移

wire [31:0] srl\_step1;

wire [31:0] srl\_step2;

assign srl\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {1'd0, alu\_src2[31:1]} // 若shf[1:0]="01",右移1位,高位补0

| {32{shf\_1\_0 == 2'b10}} & {2'd0, alu\_src2[31:2]} // 若shf[1:0]="10",右移2位,高位补0

| {32{shf\_1\_0 == 2'b11}} & {3'd0, alu\_src2[31:3]}; // 若shf[1:0]="11",右移3位,高位补0

assign srl\_step2 = {32{shf\_3\_2 == 2'b00}} & srl\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {4'd0, srl\_step1[31:4]} // 若shf[3:2]="01",第一次移位结果右移4位,高位补0

| {32{shf\_3\_2 == 2'b10}} & {8'd0, srl\_step1[31:8]} // 若shf[3:2]="10",第一次移位结果右移8位,高位补0

| {32{shf\_3\_2 == 2'b11}} & {12'd0, srl\_step1[31:12]}; // 若shf[3:2]="11",第一次移位结果右移12位,高位补0

assign srl\_result = shf[4] ? {16'd0, srl\_step2[31:16]} : srl\_step2; // 若shf[4]="1",第二次移位结果右移16位,高位补0

// 算术右移

wire [31:0] sra\_step1;

wire [31:0] sra\_step2;

assign sra\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[31], alu\_src2[31:1]} // 若shf[1:0]="01",右移1位,高位补符号位

| {32{shf\_1\_0 == 2'b10}} & {{2{alu\_src2[31]}}, alu\_src2[31:2]} // 若shf[1:0]="10",右移2位,高位补符号位

| {32{shf\_1\_0 == 2'b11}} & {{3{alu\_src2[31]}}, alu\_src2[31:3]}; // 若shf[1:0]="11",右移3位,高位补符号位

assign sra\_step2 = {32{shf\_3\_2 == 2'b00}} & sra\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {{4{sra\_step1[31]}}, sra\_step1[31:4]} // 若shf[3:2]="01",第一次移位结果右移4位,高位补符号位

| {32{shf\_3\_2 == 2'b10}} & {{8{sra\_step1[31]}}, sra\_step1[31:8]} // 若shf[3:2]="10",第一次移位结果右移8位,高位补符号位

| {32{shf\_3\_2 == 2'b11}} & {{12{sra\_step1[31]}}, sra\_step1[31:12]}; // 若shf[3:2]="11",第一次移位结果右移12位,高位补符号位

assign sra\_result = shf[4] ? {{16{sra\_step2[31]}}, sra\_step2[31:16]} : sra\_step2; // 若shf[4]="1",第二次移位结果右移16位,高位补符号位

//-----{移位器}end

// 选择相应结果输出

assign alu\_result = (alu\_add|alu\_sub) ? add\_sub\_result[31:0] :

alu\_slt ? slt\_result :

alu\_sltu ? sltu\_result :

alu\_and ? and\_result :

alu\_nor ? nor\_result :

alu\_or ? or\_result :

alu\_xor ? xor\_result :

alu\_sll ? sll\_result :

alu\_srl ? srl\_result :

alu\_sra ? sra\_result :

alu\_lui ? lui\_result :

alu\_not ? not\_result :

alu\_nand ? nand\_result :

alu\_sgt ? sgt\_result :

32'd0;

Endmodule

Alu\_display.v

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// > 文件名: alu\_display.v

// > 描述 ：ALU显示模块，调用FPGA板上的IO接口和触摸屏

// > 作者 : LOONGSON

// > 日期 : 2016-04-14

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module alu\_display(

//时钟与复位信号

input clk,

input resetn, //后缀"n"代表低电平有效

//拨码开关，用于选择输入数

input [1:0] input\_sel, //00:输入为控制信号(alu\_control)

//10:输入为源操作数1(alu\_src1)

//11:输入为源操作数2(alu\_src2)

//触摸屏相关接口，不需要更改

output lcd\_rst,

output lcd\_cs,

output lcd\_rs,

output lcd\_wr,

output lcd\_rd,

inout[15:0] lcd\_data\_io,

output lcd\_bl\_ctr,

inout ct\_int,

inout ct\_sda,

output ct\_scl,

output ct\_rstn

);

//-----{调用ALU模块}begin

reg [3:0] alu\_control; // ALU控制信号

reg [31:0] alu\_src1; // ALU操作数1

reg [31:0] alu\_src2; // ALU操作数2

wire [31:0] alu\_result; // ALU结果

alu alu\_module(

.alu\_control(alu\_control),

.alu\_src1 (alu\_src1 ),

.alu\_src2 (alu\_src2 ),

.alu\_result (alu\_result )

);

//-----{调用ALU模块}end

//---------------------{调用触摸屏模块}begin--------------------//

//-----{实例化触摸屏}begin

//此小节不需要更改

reg display\_valid;

reg [39:0] display\_name;

reg [31:0] display\_value;

wire [5 :0] display\_number;

wire input\_valid;

wire [31:0] input\_value;

lcd\_module lcd\_module(

.clk (clk ), //10Mhz

.resetn (resetn ),

//调用触摸屏的接口

.display\_valid (display\_valid ),

.display\_name (display\_name ),

.display\_value (display\_value ),

.display\_number (display\_number),

.input\_valid (input\_valid ),

.input\_value (input\_value ),

//lcd触摸屏相关接口，不需要更改

.lcd\_rst (lcd\_rst ),

.lcd\_cs (lcd\_cs ),

.lcd\_rs (lcd\_rs ),

.lcd\_wr (lcd\_wr ),

.lcd\_rd (lcd\_rd ),

.lcd\_data\_io (lcd\_data\_io ),

.lcd\_bl\_ctr (lcd\_bl\_ctr ),

.ct\_int (ct\_int ),

.ct\_sda (ct\_sda ),

.ct\_scl (ct\_scl ),

.ct\_rstn (ct\_rstn )

);

//-----{实例化触摸屏}end

//-----{从触摸屏获取输入}begin

//根据实际需要输入的数修改此小节，

//建议对每一个数的输入，编写单独一个always块

//当input\_sel为00时，表示输入数控制信号，即alu\_control

always @(posedge clk)

begin

if (!resetn)

begin

alu\_control <= 4'd0;

end

else if (input\_valid && input\_sel==2'b00)

begin

alu\_control <= input\_value[3:0];

end

end

//当input\_sel为10时，表示输入数为源操作数1，即alu\_src1

always @(posedge clk)

begin

if (!resetn)

begin

alu\_src1 <= 32'd0;

end

else if (input\_valid && input\_sel==2'b10)

begin

alu\_src1 <= input\_value;

end

end

//当input\_sel为11时，表示输入数为源操作数2，即alu\_src2

always @(posedge clk)

begin

if (!resetn)

begin

alu\_src2 <= 32'd0;

end

else if (input\_valid && input\_sel==2'b11)

begin

alu\_src2 <= input\_value;

end

end

//-----{从触摸屏获取输入}end

//-----{输出到触摸屏显示}begin

//根据需要显示的数修改此小节，

//触摸屏上共有44块显示区域，可显示44组32位数据

//44块显示区域从1开始编号，编号为1~44，

always @(posedge clk)

begin

case(display\_number)

6'd1 :

begin

display\_valid <= 1'b1;

display\_name <= "SRC\_1";

display\_value <= alu\_src1;

end

6'd2 :

begin

display\_valid <= 1'b1;

display\_name <= "SRC\_2";

display\_value <= alu\_src2;

end

6'd3 :

begin

display\_valid <= 1'b1;

display\_name <= "CONTR";

display\_value <={20'd0, alu\_control};

end

6'd4 :

begin

display\_valid <= 1'b1;

display\_name <= "RESUL";

display\_value <= alu\_result;

end

default :

begin

display\_valid <= 1'b0;

display\_name <= 40'd0;

display\_value <= 32'd0;

end

endcase

end

//-----{输出到触摸屏显示}end

//----------------------{调用触摸屏模块}end---------------------//

Endmodule

Alu.xdc

#时钟信号连接

set\_property PACKAGE\_PIN AC19 [get\_ports clk]

#脉冲开关，用于输入作为复位信号，低电平有效

set\_property PACKAGE\_PIN Y3 [get\_ports resetn]

#拨码开关连接，用于输入，依次为sw0,sw1

set\_property PACKAGE\_PIN AC21 [get\_ports input\_sel[0]]

set\_property PACKAGE\_PIN AD24 [get\_ports input\_sel[1]]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports resetn]

set\_property IOSTANDARD LVCMOS33 [get\_ports input\_sel[1]]

set\_property IOSTANDARD LVCMOS33 [get\_ports input\_sel[0]]

#触摸屏引脚连接

set\_property PACKAGE\_PIN J25 [get\_ports lcd\_rst]

set\_property PACKAGE\_PIN H18 [get\_ports lcd\_cs]

set\_property PACKAGE\_PIN K16 [get\_ports lcd\_rs]

set\_property PACKAGE\_PIN L8 [get\_ports lcd\_wr]

set\_property PACKAGE\_PIN K8 [get\_ports lcd\_rd]

set\_property PACKAGE\_PIN J15 [get\_ports lcd\_bl\_ctr]

set\_property PACKAGE\_PIN H9 [get\_ports {lcd\_data\_io[0]}]

set\_property PACKAGE\_PIN K17 [get\_ports {lcd\_data\_io[1]}]

set\_property PACKAGE\_PIN J20 [get\_ports {lcd\_data\_io[2]}]

set\_property PACKAGE\_PIN M17 [get\_ports {lcd\_data\_io[3]}]

set\_property PACKAGE\_PIN L17 [get\_ports {lcd\_data\_io[4]}]

set\_property PACKAGE\_PIN L18 [get\_ports {lcd\_data\_io[5]}]

set\_property PACKAGE\_PIN L15 [get\_ports {lcd\_data\_io[6]}]

set\_property PACKAGE\_PIN M15 [get\_ports {lcd\_data\_io[7]}]

set\_property PACKAGE\_PIN M16 [get\_ports {lcd\_data\_io[8]}]

set\_property PACKAGE\_PIN L14 [get\_ports {lcd\_data\_io[9]}]

set\_property PACKAGE\_PIN M14 [get\_ports {lcd\_data\_io[10]}]

set\_property PACKAGE\_PIN F22 [get\_ports {lcd\_data\_io[11]}]

set\_property PACKAGE\_PIN G22 [get\_ports {lcd\_data\_io[12]}]

set\_property PACKAGE\_PIN G21 [get\_ports {lcd\_data\_io[13]}]

set\_property PACKAGE\_PIN H24 [get\_ports {lcd\_data\_io[14]}]

set\_property PACKAGE\_PIN J16 [get\_ports {lcd\_data\_io[15]}]

set\_property PACKAGE\_PIN L19 [get\_ports ct\_int]

set\_property PACKAGE\_PIN J24 [get\_ports ct\_sda]

set\_property PACKAGE\_PIN H21 [get\_ports ct\_scl]

set\_property PACKAGE\_PIN G24 [get\_ports ct\_rstn]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_cs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_wr]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rd]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_bl\_ctr]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[10]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[11]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[12]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[13]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[14]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[15]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_int]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_sda]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_scl]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_rstn]